AD-A056 902 MITRE CORP BEDFORD MASS F/G 9/2
IMPLEMENTATION OF A SECURE DATA MANAGEMENT SYSTEM FOR THE SECUR--ETC(U)
JUL 78 B N WAGNER F19628-78-C-0001
UNCLASSIFIED MTR-3524 ESD-TR-78-154 NL

1 OF 1
AD
A056 902

END
DATE
FILMED
9-78
DDC

LEVEL (11)

ESD-TR-78-154 MTR-3524

AD A056902

IMPLEMENTATION OF A SECURE DATA MANAGEMENT SYSTEM FOR THE SECURE UNIX OPERATING SYSTEM

BY B. N. WAGNER

JULY 1978

Prepared for

DEPUTY FOR TECHNICAL OPERATIONS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Massachusetts

AD No.
DDC FILE COPY

DDC
RECEIVED
AUG 2 1978
F

Project No. 5720
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-78-C-0001

Approved for public release;
distribution unlimited.

78 07 27 030

When U.S. Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

William R. Price
WILLIAM R. PRICE, Captain, USAF
Technology Applications Division

[illegible]
CHARLES J. GREWE, Jr., Lt Colonel, USAF
Chief, Technology Applications Division

FOR THE COMMANDER

[illegible]
RICHARD P. RUBRECHT, Lt Colonel, USAF
Chief, Field Support Division
Directorate of Computer Systems Engineering
Deputy for Technical Operations

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

18 19 **REPORT DOCUMENTATION PAGE**

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER ESD-TR-78-154	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
6 4. TITLE (and Subtitle) IMPLEMENTATION OF A SECURE DATA MANAGEMENT SYSTEM FOR THE SECURE UNIX OPERATING SYSTEM		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER 14 MTR-3524
7. AUTHOR(s) 10 B. N./Wagner		8. CONTRACT OR GRANT NUMBER(s) 15 F19628-78-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS The MITRE Corporation P.O. Box 208 Bedford, MA 01730		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Project No. 5720
11. CONTROLLING OFFICE NAME AND ADDRESS Deputy for Technical Operations Electronic Systems Division, AFSC Hanscom Air Force Base, MA 01731		11 12. REPORT DATE JUL[redacted]78
		13. NUMBER OF PAGES 40 12 41 P.
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

INGRES
RELATIONAL DATA BASE MANAGEMENT
SECURITY
UNIX * tm (trademark)

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A secure data management system that achieves a multilevel capability for building and accessing relations has been implemented to run on the Secure UNIX Operating System for the DEC PDP-11/45. The secure DMS is an adaptation of INGRES, a relational DMS developed at the University of California, Berkeley. This paper addresses the effect of multilevel security on the design of the INGRES data base system and its user interface, including implementation details and an evaluation of the system.

78 07 27 030

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

235 050 kc

ACKNOWLEDGMENTS

This report has been prepared by The MITRE Corporation under Project No. 5720. The contract is sponsored by the Electronic Systems Division, Air Force Systems Command, Hanscom Air Force Base, Massachusetts.

ACCESSION for
NTIS White Section [✓]
DDC Buff Section ☐
UNANNOUNCED ☐
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

TABLE OF CONTENTS

ACCESSION for
NTIS
DDC
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AV
Dist.

PRECEDING PAGE BLANK

LIST OF ILLUSTRATIONS

SECTION I

INTRODUCTION

OBJECTIVE

Security kernel technology has advanced to a stage where prototypes are being built and evaluated. The addition of a data base management system to a secure operating system creates a testbed to identify and evalutate key technical issues in secure data management system technology.

Applying these concepts, the goal of this project was to choose a relational data base management system that lends itself for implementation on a secure minicomputer and to re-design such a system to incorporate Department of Defense security policies and controls. This paper addresses the implementation of a secure data base management system and the effect of kernel-enforced security constraints on the user interface. The performance of the system in the secure environment, though not a key issue, is another area of study. The results from this work will be relevant to such Air Force applications as TAC's Automated Combat Intelligence Center concept and other C3 (Communications, Command, and Control) developments.

BACKGROUND

A prototype Secure UNIX Operating System for the Digital Equipment Corporation PDP-11/45 was implemented at the MITRE Corporation under the sponsorship of ESD and DARPA Secure UNIX is the system on which the secure data management system will run. The user interface of the MITRE Secure UNIX is similar to the original UNIX Operating System, designed at Bell Telephone Laboratories for Digital Equipment Corporation computer systems (in particular, the PDP-11/40, 11/45, and 11/70). The Secure UNIX system incorporates kernel security controls that evolved from the brassboard security kernel technology. The brassboard kernel was developed to support a demonstration special-purpose tactical fusion scenario, which utilizes a primitive data management capability [1,2].

The Air Force Electronic Systems Division sponsored several research and development efforts in the design, specification, and validation of secure data base management systems. Of primary importance are the contributions from Marvin Shaefer and Thomas

Hinke of System Development Corporation and from a team at I.P. Sharp Associates Ltd. The SDC effort concerned the design of a secure relational data base management system that interfaces with the multilevel environment provided by the secure Multics Operating System. In the DMS model, although problems with the secure Multics functions for data creation, deletion, and process synchronization had not been thoroughly resolved, the designers concluded that a relational data base could comfortably exist within the multilevel environment [3]. I.P. Sharp Associates Ltd. investigated the design of kernel primitives that would support the implementation of a family of secure data management systems [4]. The resulting specifications for the primitives were proven [5] to conform to a model for a protected DMS [6]. The I.P. Sharp study recommended that security levels be assigned per relation. Both of these studies helped to identify and clarify the key technical issues in secure data base management technology.

Under ESD sponsorship, several projects attempted to evaluate the effect of security on the user interface for specific applications. In particular, the Air Force Data Services Center (AFDSC) needed a large scale multilevel secure computer that would allow users at two different security levels the ability to simultaneously access classified information. The Multics Operating System was enhanced to operate in the AFDSC's two-level (Secret and Top Secret) environment [7]. The appearance of the system to users remained almost the same as current Multics. With the addition of access levels, users needed to login at the highest access level of information in use.

The Military Message Experiment (MME), sponsored by DARPA and the Navy, is also concerned with the impact of multilevel security on a usable operator interface [8,9]. The MME project is evaluating a computer-aided message-handling system in a multilevel environment. This system is similar to a transaction-oriented data base management system. The experiment resulted in the development of a multilevel terminal to store and display information at multiple security levels. The terminal has distinct storage areas represented as windows on the display. Each window may have a separate classification, independent of the other windows.

In the current military message system, a user must be able to view information at one level, while composing or editing a message at another level. The concept of a multilevel terminal enhances the user interface by allowing a more direct interaction to occur between a user and the message system. Using the multilevel terminal, other security-related issues were evaluated such as window structure, security confirmation, trusted job interaction, and effective methods to maintain user awareness of data classification levels.

TASK APPROACH

The aim of this task is to coalesce an existing data base management system with the technology acquired from the previous studies on security kernels and secure data management systems, in order to identify and address the basic technical issues. The resulting implementation provides a means for assessing the user interface and performance in the multilevel environment.

In the field of data base management, several systems exist which could be candidates for the secure prototype DMS system. The IBM Watson Research Center's "Query-by-Example" [10], Stanford Research Institute's LADDER [11], INGRES developed at the University of California at Berkeley [12,13,14,15,16], and ZETA at the University of Toronto are sophisticated query languages suitable for a relational DMS. The INGRES (Interactive Graphics and Retrieval) System was chosen as the system to be implemented for study, because it would be the most easily adapted to run on the Secure UNIX system. INGRES is designed to run on the UNIX Operating System. The INGRES program modules are coded entirely in "C", a powerful, structured programming language, in which the UNIX Operating System is also written.

On a user level, INGRES has the advantage of being a relational system, allowing for diversified and sophisticated methods of retrieving and updating data without complex query statements. The query language of INGRES is called QUEL and consists of simple commands, sometimes terse, for accessing information in a data base. A user need not be concerned how data structures are implemented or what algorithms are operating on the stored data.

A main concern, when adapting INGRES to run in the environment provided by Secure UNIX, was the differences between the two UNIX systems that could affect the INGRES system software. To the credit of the Secure UNIX designers, the changes to the UNIX system software had a minor impact on application programs using the software. Briefly, the incompatibilities involve the removal of the "super-user" feature, non-support of the "setuid" concept, and the elimination of the inode structure. Each of these changes influenced the design of the Secure INGRES system and will be discussed in Section III of this report.

In the design and implementation of the Secure INGRES prototype system, two tasks can be identified:

a. the adaptation of INGRES to run under the Secure UNIX Operating System, assuring its compatibility in a secure environment;

b. the achievement of a true multilevel capability in building and accessing relations.

Descriptions of Secure UNIX and the INGRES DBMS are presented in Section II and Section III respectively. From these brief tutorials, a reader should have a better understanding of security in the UNIX Operating System and the concept of a relational data base system applied to INGRES. The design of the secure data base system and the user interface for the system are described in Section IV; the next section, Section V, contains a discussion of the specific modifications to the INGRES code. Concluding remarks about performance, user interaction, and an overall evaluation are presented in Section VI.

SECTION II

SECURE UNIX OPERATING SYSTEM

OVERVIEW

As stated earlier, INGRES is implemented to run on the UNIX Operating System. UNIX is a general-purpose, multi-user, interactive operating system for the DEC PDP-11 series of computers. The design of the system employs computer technology that has been used in the MULTICS and TENEX operating systems. The UNIX system is structured around a hierarchical file system, that includes compatible file, device, and interprocess input/output, initialization of asynchronous processes, and a sophisticated set of applications software [17].

The secure prototype model of UNIX is designed to incorporate security and integrity controls without altering the basic UNIX interface between system software and user application software. UNIX supports a discretionary access policy to physically protect data in its file system; additionally, the security kernel of Secure UNIX addresses data protection by enforcing a non-discretionary security policy as defined in military systems.

STRUCTURE OF SECURE UNIX FILE SYSTEM

Similar to the UNIX system, Secure UNIX also supports a tree-structured file system where each file is either a directory, data file, or special file. Special files exist for each device on the system and are defined so that operations on the device are similar to data file procedures. A directory provides the mapping between the names of files and the files themselves, thus forming an extended structure of the file system. Descriptive information about the files is also found in the directory. All files have access levels, non-discretionary (mandatory) and discretionary, which the system checks when the file is referenced. The access level of a directory is specified by a qualified user when the directory is created. The data files in a directory have the same mandatory access level as the directory.

To a user, operations on the secure file system run identically to file manipulations under UNIX. In creating a new directory, a user must adhere to the security properties established for a secure system [18]. For example, under Secure UNIX, a user can create a directory at a higher security level than he is logged in, but, in order to write into that directory, ie., to create a data file, the user must be logged in at the security level of the directory.

SECTION III

INGRES DATA BASE MANAGEMENT SYSTEM

INTRODUCTION

An important concept in the design of INGRES is that it is modeled around a relational data base. The advantages of a relational model have been discussed in the literature [19,20]. The designers of INGRES felt that a relational model would provide a high degree of data independence and a procedure-free facility for incorporating such functions as data definition, retrieval, update, access control, support of views, and integrity protection. In order for a comprehensive understanding of the design of INGRES, an introduction to a relational data base is presented, defining the structure and the functions that comprise the system.

STRUCTURE OF RELATIONAL DATA BASES

A primary consideration of a data management system is the handling of data elements or sets of elements and their inter-relationships. A relation may be represented as a table or matrix of data elements in rows and columns. Each row in a relation is called a tuple; each column consists of data elements related to a defined domain [21]. A domain has a specific format, that defines the width of the domain, meaning the number of positions for that column of information, and the type of information found in the domain, such as character or numeric.

A relationship exists between relations when tuples in one relation can be correlated with tuples in another relation. A query language is used to define the relationship and display the results of a query to the user. The purpose of a query in a relational DMS is to create a new relation containing information derived from one or more existing relations. Figure 1 illustrates a sample relationship between the relation "Officers" and the relation "Personnel Addr". The new relation called "Officers Addr" contains the results of a query to find the addresses of the officers.

Views and integrity protection are operations which enhance the functionality of a relational data base system. A view is a method of creating a relation by executing a saved set of query operations on an existing relation. The query operations are saved as the view definition. Subsequent views of a relation may be created by executing the view definition.

RELATION : OFFICERS

DOMAINS : NAME (TYPE = CHARACTER, LENGTH = 25)
RANK (TYPE = CHARACTER, LENGTH = 15)
SERIAL-NO (TYPE = NUMERIC , LENGTH < 2^{32} OR 4 BYTES)

TUPLES IN OFFICERS RELATION :

NAME	RANK	SERIAL-NO
GEORGE F. BROWN	GENERAL	975402
HAROLD W. GLENN	LIEUTENANT	132884
JAMES J. SMITH	COLONEL	546297

SAMPLE INGRES QUERY FOR CREATING RELATION AND ADDING TUPLES :

```
create officers (name = c25, rank = c15, serial no = i4)
range of o is officers
append to officers (o.name = "GEORGE F. BROWN", o.rank = "GENERAL", o.serial-no = 975402)
append to officers (o.name = "HAROLD W. GLENN", o.rank = "LIEUTENANT", o.serial-no = 132884)
```

RELATION : PERSONNEL - ADDR

DOMAINS : NAME (TYPE = CHARACTER, LENGTH = 25)
STATE (TYPE = CHARACTER , LENGTH = 4)

TUPLES IN PERSONNEL - ADDR RELATION :

NAME	STATE
ALVIN M. ALLEN	COLO
GEORGE F. BROWN	CA
HAROLD W. GLENN	MASS
JOHN D. LYONS	ALA
JAMES J. SMITH	TEX
PETER L. WILLIAMS	ILL

QUERY : MAKE A NEW RELATION CONSISTING OF ALL OFFICERS AND THEIR STATE LOCATION

COURSE OF ACTION : FIND THE TUPLES IN THE PERSONNEL-ADDR RELATION WHICH MATCH THE DOMAIN "NAME" IN THE OFFICERS RELATION. CREATE A NEW RELATION WITH DOMAINS "NAME" ,"STATE" AND "RANK" USING THE MATCHED TUPLES.

INGRES QUERY STATEMENTS FOR CREATING NEW RELATION :

```
range of o is officers
range of p is personnel-addr
retrieve into officers-addr (o.name, p.state, o.rank)
    where o.name = p.name
```

NEW RELATION : OFFICERS - ADDR

DOMAINS : NAME (TYPE = CHARACTER, LENGTH = 25)
STATE (TYPE = CHARACTER, LENGTH = 4)
RANK (TYPE = CHARACTER, LENGTH = 15)

TUPLES IN OFFICERS - ADDR RELATION :

NAME	STATE	RANK
GEORGE F. BROWN	CA	GENERAL
HAROLD W. GLENN	MASS	LIEUTENANT
JAMES J. SMITH	TEX	COLONEL

IA-51,695

Figure 1 SAMPLE RELATIONSHIP BETWEEN RELATIONS

Integrity protection involves maintaining the credibility of data in a relation. Several methods have been designed such as normal forms [22] and the assignment of integrity conditions on a set of domains. Integrity conditions may specify that a value in column 1 be greater than 0 or that the value in column 2 be equal to "Brown". Integrity protection, as applied to relational data management systems, resembles a set of consistency conditions.

DESCRIPTION OF INGRES DBMS

Under INGRES, a user has a data base consisting of relations. The data base exists as a UNIX directory created by INGRES; each relation is a UNIX file maintained by INGRES. Relations may be created by the direct input of data to the relation file or by employing INGRES queries to retrieve information from other relations.

A user is able to access the tuples and domains of an INGRES relation using the query language, QUEL, which has the capability of retrieving, inserting, and updating tuples or sets of tuples. The versatility of these operations is enhanced by the additional use of aggregate and functional expressions, that can involve arithmetic, logical, or comparison operators, over a set of tuples. Views, integrity control, and protection constraints governing relations are not supported in INGRES Version 5.1 which was used for the Secure INGRES implementation, but these primitives are expected in future versions of INGRES.

Views would be implemented similarly to the definition of views in a relational data base. Integrity control would be established by having a user enter an integrity assertion that is composed of QUEL qualification clauses to be applied to query interactions updating a relation. An example of an INGRES integrity constraint is:

```
range of p is personnel   /* declare the relation in use */
integrity constraint is
   p.name != "BROWN" and p.salary < 2000
```

In this example, further transactions on the "personnel" relation, that involve tuples whose "name" domain is "BROWN" or the "salary" domain is greater than 2000, would not be executed because of the enforcement of the integrity constraint. Protection constraints would be handled similarly to integrity control in that qualifications would be added to the user's query interactions.

When INGRES is invoked, a collection of *UNIX* processes is created and the appropriate interprocess communication is established. A process, as defined in UNIX, is an address space which has a user identifier and is the unit of work scheduled under UNIX. A process may spawn or "fork" subprocesses to execute programs. The internal design of INGRES relies on a four-process structure. The processes interact one-dimensionally as shown in Figure 2.

IA-51,691

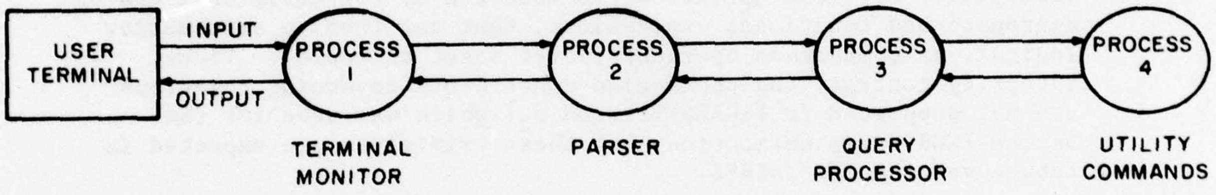

Figure 2. INGRES Process Structure

In the process structure shown in Figure 2, commands are passed to the right and retrieval information or error messages are returned to the left. Information is passed through UNIX data pipes, which are inter-process channels. The processes are synchronized so that, for example, a process must wait for a response from a process to the right before accepting information from a process to the left. Internally, specific pipes are designated as communication routes for passing data between processes. Each process knows the names of the pipes over which to send or wait for data. A file descriptor is created for each defined pipe and is the identifying name for the pipe. Communication is established by having one process read data using the file descriptor of the same pipe in which another process is writing data.

Each process is concerned with a specific task. Process 1 monitors the terminal activity of a user in building a set of INGRES queries for user execution. Process 2 translates the queries by using the YACC (Yet Another Compiler-Compiler) translator available under UNIX. A comprehensive translator is produced by utilizing the YACC tables with the parse table interpreter and locally-supplied lexical analysis. The resulting string of tokens is passed to Process 3.

In Process 3, the routines for the major INGRES commands, "retrieve", "append", "delete", and "replace", are executed. The "retrieve" command is the basic command for accessing tuples in a relation. "Retrieve" will fetch all tuples that satisfy the specified conditions given by a user and either display the tuples on the terminal or store them in a new relation. Tuples may be added to a relation by using the "append" command. Qualifying conditions may also be specified in the "append" command for selecting the tuples. Conversely, tuples are removed from a relation with the "delete" command. The "replace" command allows a user to change the values of the data elements in a specific domain, which satisfy given conditions. For example, assume a relation exists called "employee" that contains the domains "salary" and "division". If the "salary" of all employees in "division 7" needs to be increased by ten percent, the "replace" command will change the appropriate salaries.

Additional routines for the utlity commands, "create", "copy", "destroy", "help", "index", "modify", "print", "range", "save", and "sort" reside in Process 4. This process accepts the token strings of a command not executed in Process 3. Briefly, these commands involve:

a. creating a new relation (create),

b. copying data from a UNIX file into an INGRES relation (copy),

c. deleting an existing relation (destroy),

d. providing information about how to use INGRES (help),

e. creating a secondary index on an existing relation to make retrieval and update operations on the relation more efficient (index),

f. converting the storage structure of a relation (modify),

g. printing the contents of a relation (print),

h. declaring a relation and its variable for use in a query (range),

i. saving a relation until a certain date (save), and

j. sorting a relation into ascending order (sort).

Certain commands in INGRES may be executed only from the UNIX shell, thus not requiring the INGRES monitor. INGRES data bases are created and deleted from the UNIX file structure using, respectively, the commands "creatdb" and "destroydb". The command "ingres" invokes the data management system. Data for the "copy" utility may be formatted using the "format" command. The contents of a relation are printed without having to enter INGRES by executing the "printr" command. Lastly, if an INGRES or UNIX system crash occurs, the "restore" command removes any extraneous information from INGRES data bases.

The INGRES file structure can be represented as a sub-tree of the UNIX system. Six directories, which descend from a special root directory owned by the UNIX user "ingres", form the main structure. Figure 3 illustrates this file structure. The contents of each directory will be briefly described. The directory named "files" contains initialization files for spawning the four INGRES processes and other INGRES command information needed by the system. An "authority" file in this directory exists for maintaining a list of users who are allowed to create INGRES data bases.

All data bases are created off the "datadir" directory. Essentially, an INGRES data base takes the form of a UNIX directory. Descending from a data base directory are the system relation files, used by the INGRES processes, and the user relations files created by either the owner of the data base or other users. Each relation is created as a separate UNIX file. Under INGRES, relations created by the data base owner can be accessed by any other user; other user relations are not shared. An administration data file also exists which contains data base initialization information.

The "bin" and "source" directories are comprised, respectively, of the binary and source code files. The libraries needed for developing new binary modules are found in the "lib" directory. The "tmp" directory has data files designated for use as workspace areas with the terminal monitor process. The complete set of documentation for INGRES is contained in the "doc" directory.

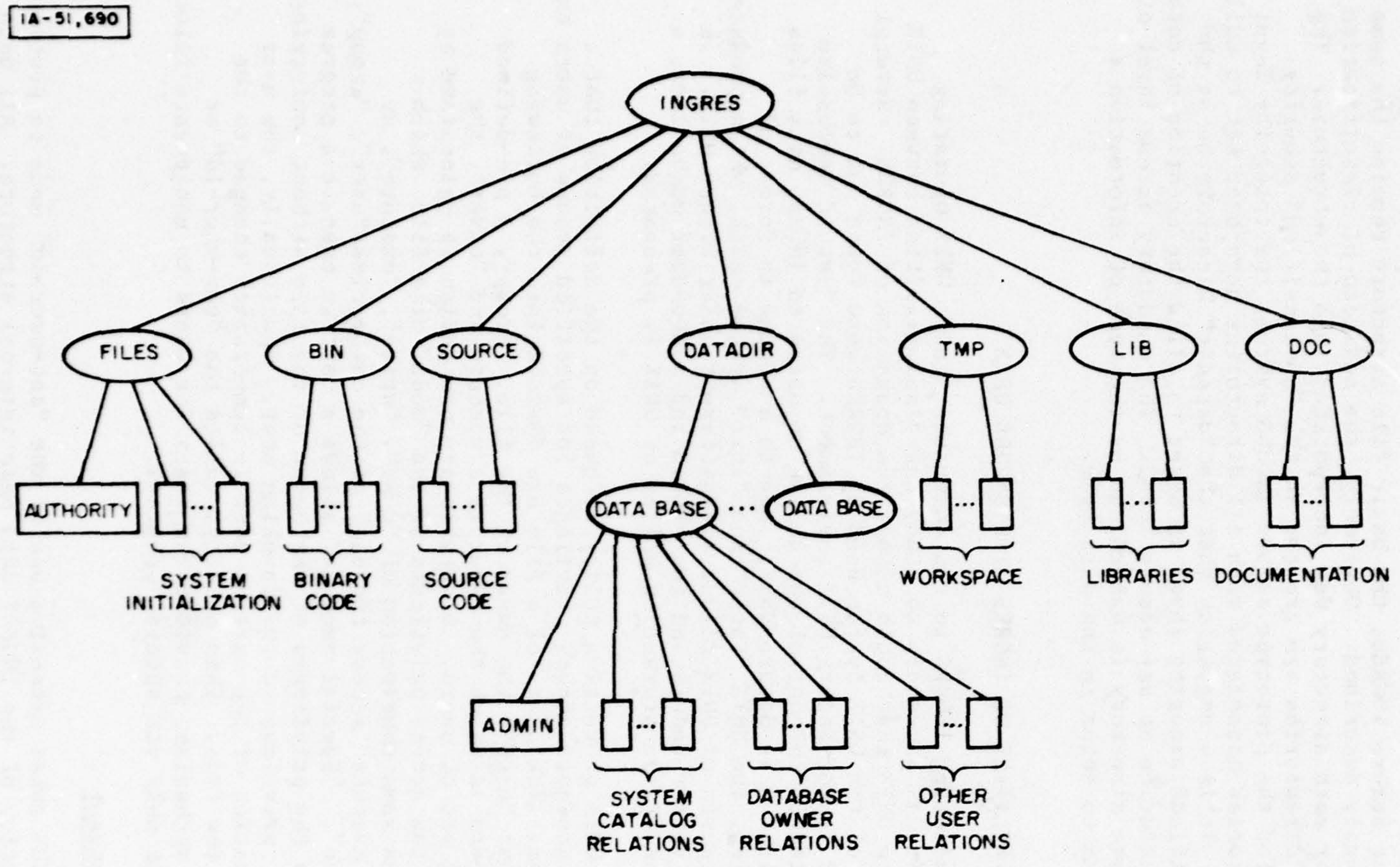

Figure 3 INGRES FILE STRUCTURE LAYOUT

Under Secure INGRES, the basic file structure remains the same as previously described. Obviously, the addition of classification levels for each directory was incorporated into the structure. The six main directories are created at the "unclassified" security level. For the prototype secure INGRES system, the integrity level and categories associated with all directories have been set to null or zero. It is a necessity that the "datadir" directory be at the "unclassified" security level in order to allow the creation of data base directories at any access level. The mandatory access level of a data base directory is determined by the type of information a user plans to enter in the data base.

INCOMPATIBILITIES OF INGRES WITH SECURE UNIX

In adapting INGRES to run under the Secure UNIX Operating System, it was necessary to study the dissimilarities between UNIX and Secure UNIX that could affect the operation of INGRES. Several features of the UNIX system used by INGRES were found not to be supported in the Secure UNIX environment. The "setuid" mechanism used by INGRES to control the access of users to INGRES data files was eliminated in Secure UNIX. Due to a change in format of directories, the UNIX "fstat" and "stat" system calls, which produce status reports of UNIX files, are different under Secure UNIX. In order to better understand the setuid and super-user mechanisms, a discussion of the protection policy of UNIX is presented.

The UNIX protection policy is based on the definition that a file has assigned access privileges for specified groups of users on the system. The users of a file are divided into the following categories: "user", the owner of the file, "group", a pre-defined set of users in which the owner is a member, and "other", the remaining set of users. Discretionary protection is maintained by checking the access privileges of the "mode" of a file, which designates some combination of "read", "write", "execute", or "special execute" access for each of the categories "user", "group", and "other". "Special execute" allows a user to execute a program file with the privileges of the owner of the file without conferring arbitrary privilege to the invoking user. Specifically, the user identification of the current user is temporarily changed to the owner of the file. This ability, called the "set-user-Id" or "setuid" mechanism, provides the means for users to manipulate files accessible only via special programs.

Setuid Concept

INGRES makes extensive use of the "set-user-Id" mode to protect the integrity of the INGRES data base internal structure. All data files accessed by the INGRES system software are owned by the UNIX user "ingres". The discretionary protection mode for these files is "read-write for owner, no other access". INGRES commands are also owned by the user "ingres" with the mode of the command files set to "special execute, no other access". Users executing INGRES commands, in effect, assume the access privileges of the user "ingres" until the command is completed. Under UNIX, the system call "setuid" is invoked so that INGRES processes are operating temporarily with the user-Id of "ingres".

In the Secure UNIX environment, the "setuid" and "setgid" system calls are not implemented because they can permit a compromise of security. With the setuid feature, a user process is able to obtain information that it might not be allowed to access, if its real user/group identity was checked for discretionary privilege. This same restriction is applied to the "super-user" concept. A super-user can circumvent all access controls designated by users for their files. He can even change the owner of a file. Security contraints would be violated under these conditions; therefore, the super-user and setuid mechanisms are not supported.

To create an INGRES module that can be run without the setuid feature, the access privileges of INGRES object and data files were changed to reflect the privileges for a "group" of INGRES users. A user must be in the same group in which the user "ingres" belongs in order to run the complete set of INGRES commands. In effect, INGRES data files have "read-write" permission for the owner of the file and any users in the owner's group. Other users of the system, not in the "ingres" group, can execute INGRES commands that require only "read" access of INGRES data files.

The modes of the object files, which have the "special execute" designation for the setuid feature, were changed to give "read-write-execute" access to "ingres", the owner of the file, and "read-execute" or "execute-only" access to the "group" and "other" users of the program file. The new modes for the INGRES object files are listed in Figure 4.

System Calls

The other area of incompatibility involves the different format for a directory file entry. Two system calls, "stat" and "fstat", reference the information in a directory and are dependent on its format. Under Secure UNIX, the system calls return additional information into a buffer area defining the status of a file. The inode number of the file has been replaced by a unique identifier because, under Secure UNIX, the inode table is non-existent. The source code for two INGRES programs, "access/batch.c" and

INGRES FILE NAME	DISCRETIONARY ACCESS MODES FOR		
	OWNER	GROUP	OTHER
CREATDB	RWX	X	X
DESTROYDB	RWX	X	X
INGRES	RWX	RX	RX
OVERLAY	RWX	X	X
OVERLAY B	RWX	X	X
OVERLAY E	RWX	X	X
OVERLAY F	RWX	X	X
OVERLAY N	RWX	X	X
OVERLAY R	RWX	X	X
OVERLAY X	RWX	X	X
PARSER	RWX	X	X
PRINTR	RWX	X	X
QRYPROC	RWX	X	X
RESETDB	RWX	RX	RX
RESTORE	RWX	RX	RX

LEGEND

R = READ ACCESS
W = WRITE ACCESS
X = EXECUTE ACCESS

IA-51,692

Figure 4 DISCRETIONARY ACCESS MODES FOR SECURE INGRES OBJECT FILES

"access/heapsize.c", was changed to adjust for the different directory structure.

SECTION IV

SECURITY LEVELS FOR THE INGRES DBMS

INTRODUCTION

The imposition of non-discretionary (mandatory) security constraints on an INGRES data base is the main focus of this task. In the Secure UNIX environment, a non-discretionary access level, composed of security and integrity constraints, is required by the security kernel for all files. The security kernel applies the prinicples of the "*-property" for secure computer systems [18] in validating user requests of files at various access levels. Concerning directory file creation, the "*-property" concept allows a user to create a directory only at an access level equal to or greater than the current level at which the user is logged in. Mandatory and discretionary access checking is always performed when a file is created or accessed.

The design of the Secure UNIX file system requires that data files in a directory assume the same mandatory access level as the directory. Consequently, the security level of an INGRES relation (a data file) must be at the same level as its data base (a directory). Although this limitation does reduce the utility of the secure data base system, the coordination of INGRES with Secure UNIX necessitates the mapping of relations in this manner. Adhering to these restrictions, a user is able to perform multilevel operations on relations in the INGRES data base directory structure.

DESIGN OF SECURE INGRES DATA BASE SYSTEM

In re-designing INGRES to incorporate security levels, it was necessary to introduce two capabilities into the system:

a. the ability to create data bases at different security levels, and

b. the ability to create and retrieve relations at different security levels.

If a user wants to create or retrieve a relation consisting of tuples from relations at different security levels, certain restrictions under Secure UNIX limit the method of operation. Foremost, concerning access levels, the files in a directory assume the same access level as the directory. Similarly, INGRES relations in a data base are required to have the same classification as the data base. For example, when operating in a secure environment, a relation containing "top secret" information cannot be found in a data base that is classified as "confidential". As a result of these security restrictions, a user must create a data base at each security level of the information in use.

Applying these same security principles, a user is able to "read" information from a file at a lower security level than his current level established at login time. As a result, a new relation can be created by combining information (tuples) obtained from relations in data bases at access levels equal to or less than the security level of the data base in use.

The design modifications for creating a secure data base directory were easily adapted into the INGRES system due to the dependency on the Secure UNIX design, which allows the creation of directories at different security levels. With the addition of security levels, the INGRES system directories, which are separate from the "datadir" branch of the INGRES sub-tree, must be set to the lowest classification level of the system so that the system programs in these directories can be accessed/executed by users operating in data bases of any security level.

USER INTERFACE

The term "user interface" implies the appearance of a system that is presented to a user. It involves anything a user types or sees on a terminal.

Overview

The Air Force Data Services Center Multics and the Military Message Experiment are two examples of projects that have spent a considerable amount of time and effort in evaluating the effect of security on user interfaces. Pertaining to relational data base systems, System Development Corporation did a study of a secure DMS for the secure Multics system [3]. To incorporate non-discretionary security into the data base, the domains of a relation were defined to have different security levels. Each field of data in the relation is stored in separate segments at the classificaton level of the domain. The study mentioned that for certain functions such as the creation of segments, a user process would be required to operate at all levels at which data is stored. In order for a practical user interface with multilevel domains in a relation, the study suggested that a multilevel daemon process be added to the operating system. The daemon process would handle the functions that operate on multilevel domains of data.

Contrarily, the I.P. Sharp Associates Ltd. study recommended that security levels be assigned to individual relations. Because protection levels associated with tuples or domains can cause restrictions on query activity, the relation assignment proved to involve less problems while still preventing unauthorized access [6].

Secure INGRES Requirements

In Secure INGRES, relations assume the access level of the data base in which they exist. This restriction is necessary due to the structure of the secure file system under Secure UNIX. Concerning the user interface, the addition of security levels and the adjustments for Secure UNIX compatibility are relatively transparent to a user. It was necessary to change the format of two INGRES commands to accommodate security levels.

The command for creating a new data base, "creatdb", was modified to include an access level for data bases at higher security levels than the current level. The "-a" option is used if the access level of the data base should be created at the current level of user login. The format of the command is:

```
creatdb "database-name" {"access-level" or -a }
```

The "database-name" is limited to fourteen characters in length. The "access-level" is comprised of an integrity level, a security level, an integrity category, and a security category [23]. The principles of the "*-property" form the basis for access checking by the security kernel on the "access-level" entered for this command. User input for the different parts of the "access-level" is highly dependent on the security procedures of the installation using Secure INGRES and on the rules of the kernel. Secure UNIX supports under its mandatory security policy sixteen security classifications and sixty-four security categories. The mandatory integrity policy is developed from a maximum of sixteen integrity classifications and eight integrity categories.

In an actual operation environment, the "access-level" would probably consist of specific characters, which represent the different security classifications and categories. A system routine would look up the characters in a pre-defined symbol table and translate the characters into the appropriate "bit" representation required for the access structure under Secure UNIX. For the Secure INGRES prototype system, the classification and categories are represented by integer numbers, whose format is determined by the structure of the access level. The security and integrity levels are each represented by a number not greater than 15; the smaller the number, the lower is the classification or integrity level. The integrity category is represented by an integer number not greater than 255, because the size of the category is limited to eight bits, as defined in the access structure. Four numbers are necessary to represent the security category. The numbers chosen to represent the integrity and security categories have no particular significance in relation to a protection policy. For any future Secure INGRES applications on Secure UNIX, the priorities and degree of protection would be established, and the elements which comprise the access level would be more realistically defined.

The second command affected by the security changes is the "range" statement. When INGRES is invoked, the command is used to declare variables that are referenced in subsequent INGRES queries. In the "range" statement, a user defines a variable to be associated with a specific relation; thereafter, all query references to that variable involve the tuples in the associated relation. The format of the "range" statement is:

```
range of "variable" is "relation-name"
```

If a user wants to access a relation that is not in the current data base, he must start the "relation-name" with a slash, "/", followed by the data base name in which the relation resides, followed by another slash, and then the name of the relation. Examples of the two types of range statements follow:

```
range of var1 is rel1
range of var2 is /dbname/rel2
```

Secure UNIX performs the necessary mandatory and discretionary access checks when the relation file is opened for query processing.

To better illustrate how the above changes affect the interaction of a user at a terminal, a sample user session is presented involving these changes. Assume a user has logged in at the "secret" classification level and invokes INGRES using the data base "secr", which is defined at a "secret" classification level. The user wants to build a new relation using information from a relation named "type" in the "secr" data base. However, he would also like to access data from a relation named "acro", which is an "unclassified" relation in the "unclassifed" data base named "uncl". Figure 5 shows the configuration of data for the relations. The "acro" relation has two domains named "id" and "acro"; similarly, the "type" relation has "id" and "type" domains. The purpose of the query is to find the tuples in the two relations that have the same "id" and select from those matched pairs of tuples a subset of tuples where the "type" domain is equal to "orgz". The resulting tuples are placed in a new "secret" relation called "orgz". Only the domains "id" and "acro" are defined for the tuples in this relation. A user would enter the following INGRES commands to achieve the described results:

```
     range of r1 is /uncl/acro          /* declare that the relation */
                                        /* is not in the current db */
     range of r2 is type                /* relation in current db */
     retrieve into orgz(r1.id,r1.acro)  /* specify new relation */
                                        /* and its domains        */
        where r1.id = r2.id             /* specify conditions     */
           and r2.type = "orgz"
```

UNCLASSIFIED (U) RELATION ACRO

IN DATABASE UNCL (U)

DOMAINS

	ID	ACRO
TUPLES	1	ESD
	2	UHF
	3	AFSC
	4	AWACS
	5	TDMA
	6	SATIN

SECRET (S) RELATION TYPE

IN CURRENT DATABASE SECR (S)

DOMAINS

	ID	TYPE
TUPLES	1	ORGZ
	2	TECH
	3	ORGZ
	4	PROJ
	5	TECH
	6	PROJ

RESULT:

NEW SECRET (S) RELATION ORGZ

IN CURRENT DATABASE SECR (S)

DOMAINS

	ID	ACRO
TUPLES	1	ESD
	3	AFSC

IA-51,693

Figure 5 EXAMPLE OF CREATING "SECRET" RELATION FROM "SECRET" AND "UNCLASSIFIED" RELATIONS

SECTION V

IMPLEMENTATION DETAILS

IMPLEMENTATION OVERVIEW

The implementation of a secure INGRES data base management system proved to be a task devoted more to the understanding of how INGRES works than the re-coding of INGRES programs. All data base systems have a tendency to be large and cumbersome, and INGRES is no exception. The four-process structure of INGRES may contribute to a sense of logical program flow of the system, but, in effect, one process can not execute the entire set of INGRES program modules. In Version 5.1 of INGRES, which was used to develop the prototype model, a process runs in approximately 64K bytes of core memory. The INGRES system needs four processes, or the equivalent amount of memory space, in order to run the binary program modules. The total size of the INGRES DBMS is close to 275K bytes of memory. The code for the process running the utility commands is divided into overlay programs in order to conserve execution space. Future versions of INGRES need five processes to run the code. Besides the INGRES query modules which run under the INGRES monitor, code exists for six INGRES commands that are executable only from the UNIX shell. These commands range in size from 23K bytes for the "creatdb" program to 3K bytes for the "format" code. The Secure INGRES modules had relatively few code changes; thus the new program sizes of Secure INGRES vary only slightly compared to INGRES modules. The exact sizes of the program modules for both INGRES and Secure INGRES are found in Figure 6.

The documentation, provided with INGRES, was limited in its description of the many routines and subroutines and their relationship to the process structure of INGRES. Additional program details would have been beneficial in tracing the effects of modifying the "range" statement in INGRES. The INGRES code for version 5.1 is unevenly written, even though the source programs are coded extensively in "C", which is a highly structured language. A new version of INGRES (6.0) has been released, which, purportedly, has better documentation; the source code has also been completely re-written in a more standardized manner.

MODIFICATIONS TO INGRES CODE

A detailed description of the changes to the INGRES code is presented so that Secure INGRES data management systems can be

		INGRES	SECURE INGRES
PROGRAM NAME	FUNCTION	SIZE IN BYTES	SIZE IN BYTES
CREATDB*	COMMAND TO CREATE AN INGRES DATA BASE	18,968	4,988
DESTROYDB*	COMMAND TO DELETE AN INGRES DATA BASE	7,984	7,640
EQUEL	LANGUAGE PROCESSOR TO HANDLE QUEL STATEMENTS EMBEDDED IN "C" LANGUAGE	28,870	28,870
FORMAT*	COMMAND TO CONVERT FREE FORMAT FILE TO FIXED FORMAT FOR "COPY" UTILITY	3,440	3,440
INGRES*	COMMAND TO INVOKE INGRES DBMS	9,370	22,680
OVERLAY	CONTROLS OVERLAYING OF UTILITY PROGRAMS	9,854	9,854
OVERLAYB	OVERLAYS UTILITY COMMANDS "HELP" AND "PRINT"	24,772	25,184
OVERLAYE	OVERLAYS UTILITY COMMAND "COPY"	29,314	29,790
OVERLAYF	OVERLAYS UTILITY COMMAND "INDEX" AND BATCH UPDATE MODULE	27,824	28,236
OVERLAYN	OVERLAYS NULL COMMAND	9,862	9,862
OVERLAYR	OVERLAYS UTILITY COMMAND "SAVE"	21,772	22,248
OVERLAYX	OVERLAYS UTILITY COMMANDS "CREATE","DESTROY","MODIFY"	46,900	47,312
PARSER	MODULE USED BY PROCESS 2 TO HANDLE PARSING	49,994	50,176
PRINTR*	COMMAND TO PRINT CONTENTS OF RELATION	17,930	17,848
QRY PROC	MODULE USED BY PROCESS 3 FOR QUERY PROCESSING	55,532	56,812
RESTORE*	COMMAND TO RESTORE DATA BASES AFTER CRASH	23,328	23,522
	TOTAL SIZES	385,714	388,462

*DESIGNATES PROGRAM IS EXECUTABLE ONLY FROM THE UNIX SHELL

IA-51,694

Figure 6 PROGRAM SIZES OF INGRES AND SECURE INGRES

easily constructed in the future. The changes to the INGRES code can be divided into four areas. The code was impacted by:

a. removing the "setuid" feature,

b. changing the format of a directory entry,

c. adding security levels, and

d. allowing a user to query a relation not residing in the current data base.

When the "setuid" feature was eliminated, the discretionary access permissions of certain INGRES files had to be adjusted to allow users in the "ingres" group and other users of the system the ability to access these files, necessary for running under the INGRES system. In general, the user "ingres" and members of the "ingres" group have "read-write" access to most data files; other users have only "read" access to these files. Specifically, in a data base directory, the system catalog files and any other relations, created by the user "ingres" or users in the "ingres" group, have "read-write" access. The "admin" file in the data base has "read-write" access for the owner of the data base and only "read" access for the owner's group and any other users. In the INGRES source code, when these files are created, the mode used in the "creat" system call was changed to reflect the new access privileges. The following source programs were modified: "support/creatdb.c", "support/ccreate.c", "dbu/create.c", and "dbu/hash.c".

Two system files, "files/authority" and "files/cdbi", also had their discretionary mode changed: the "authority" file had "read" access extended to users in the "ingres" group, and for the "cdbi" file, "read" access was given to both users in the "ingres" group and other users. The "cdbi" file, which contains the formats of all INGRES structures, is used in the initialization of the system catalogs in a data base. The "setuid" and "setgid" system calls were removed from the programs: "support/creatdb.c" and "support/ingres.c".

The other incompatibility, which concerns a Secure UNIX directory entry containing different information than a UNIX entry, affected INGRES only in circumstances when a reference was made to the elements in the directory entry. Two system calls, "fstat" and "stat", give the status of a file by copying the information found in the directory into a buffer area. These calls are used in INGRES programs, "access/heapsize.c" and "access/batch.c", and the buffer structures defined in the programs had to be adjusted to account for the twenty-three word buffer size of Secure UNIX instead of using the eighteen word buffer declared in UNIX. The "access/batch.c" program accesses a particular element of the buffer, the inode number, and, because inodes are no longer supported by Secure UNIX, a portion of the code had to be re-written using the new information in the entry to achieve the correct results.

The addition of security levels affects, in an obvious sense, the user input when creating a data base. Because all data bases are directories under Secure UNIX, a mandatory access level is necessary when a directory is created. The "support/creatdb.c" program, which creates a data base, was modified to accept the additional user input specifying that a data base be created either at the user's current access level or at a higher level. The program interprets the input and executes the system call, "mknod", with the appropriate parameters, to create the directory for the data base. The source code was simplified by removing the unnecessary spawning of a child process to run the "mkdir" user command, which executes a "mknod" system call. The "support/destroydb.c" program, which deletes a data base, was also modified, similarly, by replacing the user command "rmdir" with a direct call to the system procedure "unlink". It became apparent that other enhancements to the INGRES code could have been made, which probably would affect execution speed, but it was not the intended goal of the project to analyze all the INGRES modules for code improvements.

The initialization of the system catalogs for a data base is also executed in the "support/creatdb.c" program. If a data base is created at a higher access level than the user login level, security policy prohibits any writing into the directory, ie., creating a file in the directory, until a user is logged in at the higher level. In INGRES, the system catalog files are established when the data base directory is created. If this same procedure was followed in Secure INGRES, the initialization of the system catalogs for a data base directory created at a higher access level would result in a security violation. To solve the conflict, the source code for creating the initialization and system catalog files of a data base, was moved to the program "support/ingres.c".

The "support/ingres.c" program contains the source code for the INGRES command, "ingres", which is executed from the UNIX shell to invoke the INGRES data base management system. In order for a user to create a relation in a data base, he issues the command "ingres" with a data base name. With the revised program code of "ingres", the first user of a new data base must be the owner of the data base. This procedure is required so that the system files of the data base are initially established with the correct format, ie. the system files must have the same owner as the data base. In addition, the owner of the data base must be logged in at the proper access level for writing in the data base. Other users will be denied access to the data base until the system files are initialized.

Basically, the code changes to the "ingres" program involve first checking if the system files exist by trying to open a known system file called "attribute". If the "open" fails, the program assumes the system catalogs have not been created and proceeds to check if the user of the INGRES process is the owner of the data base. If it is not the owner, the program returns to Secure UNIX, notifying the user of the error; otherwise, the system files are created using the code that was previously in "support/creatdb.c" for creating catalogs, and the "ingres" program continues with the spawning of the four INGRES processes. If the system catalog "attribute" is opened, the "ingres" program assumes the system files have already been created; the "attribute" file is then "closed" to restore the system to its initial state and execution resumes with the spawning of INGRES processes.

When a user wants to query a relation that is not in the current data base, he formats the "range" statement of INGRES to include the data base name as part of the relation name. The data base name is prefaced with a slash character and separated from the relation name with another slash character. In INGRES, the program "parser/range.c" stores the entire name into a range table. It is necessary to have the entire name in the table to designate that the relation exists in another data base. At this stage, the code changes to INGRES programs involve defining a new constant called "RANGNM" equal to thirty-one characters, which represents the maximum number of characters allowed for a relation name, including the data base name, in the range statement. The context file "ingres.h" contains the new constant definition; in the context file "parser.h", the structure for the range table is defined and the length of the element, "relnm", in the structure must be changed to "RANGNM". The program "parser/range.c" has routines which insert the relation name into the range table and pass the name to the query processor. Each reference to the length of the relation name must be changed to the new length, "RANGNM".

The query processor uses two structures, "rangv" and "lrangv", which contain references to the name of a relation. The structures are defined in "qryproc/qryproc.h", and the length of the relation name in these structures must be changed to "RANGNM".

When query processing is initiated, descriptive information about an "opened" relation is stored in the sub-structure of the "rangv" structure, called "descriptor". The "descriptor" structure is filled with information obtained from the system catalogs, which maintain information about all relations in the data base. A relation is opened and its descriptor established during the execution of the source program "access/openr.c".

One argument passed to the "openr" routine is a relation name, which is retrieved from the range table. With the Secure INGRES changes, the relation name may include a data base name. The argument must be re-defined so that its length is equal to "RANGNM". The previously defined argument "uniqid", is used as a local variable to represent the name of the relation without the data base name.

The source code of the "openr" program was modified to account for a relation not in the current data base. A flag is set if the data base is part of the relation name, and the relation name is parsed into character strings containing the data base name and the single relation name. The descriptor structure is then stored with information about the relation obtained from the system catalogs. The data base flag is always checked to determine if information is retrieved from the catalogs in the current data base or a different data base directory. Error messages are returned to a user if information in a directory cannot be accessed. Lastly, the "openr" routine opens the relation file and its file pointer is recorded in the destriptor structure.

The basis of all query processing is the descriptor structure which specifies how to access the tuples of a relation. Using the descriptor for subsequent relation operations, it becomes unnecessary to reference the system catalogs each time a tuple in a relation is accessed; the descriptor structure for each relation contains the appropriate information.

During query processing, a "strategy" routine is called in an attempt to limit the scan of a relation by determining a key, which is used for subsequent access calls on the relation. Information in the relation descriptor structure is used to help construct the key. If the relation has been previously set up with keys by a user executing the "index" command, the strategy routine uses the "indexed" version of the relation. Information about the indexed relation is found in the system catalog relation called "index". If the relation is not in the current data base, the strategy routine must use the "index" catalog in the appropriate data base to retrieve the correct information about the relation.

The source program of the strategy routine, "qryproc/strategy.c", was modified to operate with relations not in the current data base. The "Sourcevar" element in the range table specifies which relation is currently being accessed. If the "relation id" in the range table has a data base name as part of its relation name, then the appropriate "index" system catalog for the data base is opened, and its file pointer is stored in the global variable "Sysindex". Later in the program, if the strategy routine finds that a relation is indexed, then the correct "index" system catalog is referenced using the variable "Sysindex". When the strategy key has been determined, the new information about the relation is recorded in the descriptor named "Scanr", which keeps track of the relation being scanned. If the "Sysindex" variable has been changed, it is restored to the file pointer of the "index" catalog in the current data base.

Changes were also made to the "exactkey" and "rangekey" routines in the strategy program. The second argument of these routines, the relation name, may include a data base name, if the relation for scanning is not in the current data base. The "xxsets" routine in the source file "qryproc/strat2.c" is called by both of the keying routines and uses the second argument, the relation name, to update the global variable "Scanr". Modifications were made to this routine to adjust for a relation name that includes a data base name. The argument is compared to the "Source" relation name; if the data base name is part of the argument, the "Source" name must be prefaced with the current data base name in order for a correct comparison.

SECTION VI

CONCLUSION

An existing data base management system has been modified to incorporate security controls, as defined by DoD security policies, with the intention of operating the system on a secure mini-computer. The resulting product, a Secure INGRES relational data base management system, can be executed under the Secure UNIX Operating System and forms a basis for secure user interface analysis and user application work plus aiding in performance studies on the secure operating system.

At this stage, the Secure INGRES implementation is not operational. The programs for the Secure INGRES model have been compiled, and the new system is theoretically ready to run. Modifications to the INGRES DBMS to allow execution in the secure environment were made without extensive programming changes. Two factors contributed to the relative ease in converting the system:

a. the INGRES DBMS is designed to run under the UNIX Operating System, and

b. only a few incompatibilities exist between the Secure UNIX Operating System and the UNIX Operating System that affect the execution of the INGRES system software.

Testing of the system has been delayed because the Secure UNIX system is not yet operative. Once Secure UNIX is running, Secure INGRES can be tested and completely debugged. At that time, the issues of user interface and performance can be more thoroughly addressed.

In retrospect, considerable time was spent in an analysis of the relationship between the INGRES program modules. The lack of adequate documentation and the poorly written source code made this task more arduous than expected. Once a comprehensive knowledge of INGRES was acquired, the task of re-designing INGRES for a secure environment went smoothly and without difficulty.

The present user interface of INGRES could be improved in many ways. On a purely functional level, INGRES should supply 1) a prompt character, 2) better error messages to the user, 3) the rubout character should be handled properly, and 4) a user should be able to use the UNIX shell while executing under INGRES. A major complaint about INGRES is the amount of time required to process a query. Performance would probably improve if 1) batch-updating for relations was removed resulting in query modifications being done directly, 2) only one secondary index being allowed, 3) only heap and hash storage files being supported, not ISAM, and 4) a more efficient use of the "C" language being employed. In a specific application environment, a better user interface could be achieved by developing a set of pre-canned "C" routines, that would internally access QUEL programs. The routines would be designed so that a user could easily create queries defined for a particular application.

Users of the Secure INGRES system must be acquainted with the security policies enforced under the Secure UNIX system in order that the operation of a user session run as smoothly as when executing under INGRES. The access level of each Secure INGRES data base must be remembered by a user when retrieving information from a data base; otherwise, security measures may be violated resulting in a disqualified data query. Overall, the changes in definition and format of certain INGRES commands are minor; once a user becomes accustomed to the Secure UNIX file system and, if he has a working knowledge of INGRES, operating under Secure INGRES should be a simple adjustment. Any user applications, developed under INGRES, can also be run under Secure INGRES with only minor changes to account for access levels.

The performance of the Secure INGRES system is obviously affected, to some degree, by the additional security checks and secure system controls that are necessary for a secure operating environment. At this time, no detailed performance measurements have been made. Considering the degree of modification and the areas involved in the INGRES code, the effect of the changes to the code upon the performance of Secure INGRES should be relatively insignificant. Running in the secure environment is the major factor that affects the evaluation of Secure INGRES performance.

The Secure INGRES data base management system is an effective, software tool for diversified data manipulation of sensitive information requiring security protection. It can also provide a means for developing workbench programs to be used in performance testing of secure operating systems. With future improvements to INGRES system software, a Secure INGRES system becomes a productive addition to a secure operating system.

REFERENCES

1. W.L. Schiller, "The Design and Specification of a Security Kernel for the PDP-1145", ESD-TR-75-69, Electronic Systems Division, AFSC, Hanscom AFB, Massachusetts, May 1975 (ADA011712).

2. J.L. Mack and B.N. Wagner, "Secure Multilevel Data Base System: Demonstration Scenarios", ESD-TR-76-158, Electronic Systems Division, AFSC, Hanscom AFB, Massachusetts, October 1976 (ADA032956).

3. Thomas H. Hinke and Marvin Shaefer, "Secure Data Management System", RADC-TR-75-266, System Development Corporation, Santa Monica, California, November 1975.

4. Gillian Kirkby and Michael Grohn, "On Specifying the Functional Design for a Protected DMS Tool", ESD-TR-77-140, I.P. Sharp Associates Ltd., Ottawa, Canada, March 1977.

5. Gillian Kirkby and Michael Grohn, "Validation of the Protected DMS Specifications", ESD-TR-77-141, I.P. Sharp Associates Ltd., Ottawa, Canada, April 1977.

6. Michael J. Grohn, "A Model of a Protected Data Management System", ESD-TR-76-289, I.P. Sharp Associates Ltd., Ottawa, Canada, June 1976.

7. J.C. Whitman, A. Bensoussan, P.A. Green, A.M. Kobzian, and J.A. Stern, "Design for Multics Security Enhancements", ESD-TR-74-176, Honeywell Information Systems, 1974.

8. Stanley R. Ames, Jr., "User Interface Multilevel Security Issues in a Transaction-Orientated Data Base Management System", MTP-178, The MITRE Corporation, Bedford, Massachusetts, December 1976.

9. J.D. Tangney, S.R. Ames, Jr., and E.L. Burke, "Security Evaluation Criteria for MME Message Service Selection", MTR-3433, The MITRE Corporation, Bedford, Massachusetts, June 1977.

10. Moshe M. Zloof, "Query-by-Example", Proc. 1975 National Computer Conference, AFIPS Press, May 1975.

REFERENCES (Continued)

11. Earl D. Sacerdoti, "Language Access to Distributed Data With Error Recovery", Stanford Research Institute, Artificial Intelligence Center, Menlo Park, California, February 1977.

12. G.D. Held, M. Stonebraker, and E. Wong, "INGRES - A Relational Data Base Management System", Proc. 1975 National Computer Conference, AFIPS Press, 1975.

13. William Zook et. al., "INGRES - Reference Manual (Version 5)", University of California at Berkeley, Electronics Research Laboratory, Memo. No. ERL-M585, April 1976.

14. M. Stonebraker and E. Wong, "Access Control in a Relational Data Base Management System by Query Modification", Proc. 1974 ACM National Conference, San Diego, California, November 1974.

15. M. Stonebraker and P. Rubenstein, "The INGRES Protection System", Proc. 1976 ACM National Conference, Houston, Texas, October 1976.

16. M. Stonebraker, E. Wong, P. Kregs, and G. Held, "The Design and Implementation of INGRES", INGRES Documentation from University of California at Berkeley, December 5, 1975.

17. Dennis M. Ritchie and Ken Thompson, "The UNIX Time-Sharing System", Communications of the ACM, Volume 17, Number 7, July 1974, pp.365-375.

18. D.E. Bell and L.J. LaPadula, "Secure Computer Systems", ESD-TR-73-278, Volumes I-III, Electronic Systems Division, AFSC, Hanscom AFB, Massachusetts, November 1973-April 1974 (AD770768, AD771543, AD780528).

19. E.F. Codd, "A Relational Model of Data for Large Shared Data Banks", Communications of the ACM, Volume 13, Number 6, June 1970.

20. E.F. Codd and C.J. Date, "Interactive Support for Non-Programmers, The Relational and Network Approaches", Proc. 1974 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, Michigan, May 1974.

21. E.F. Codd, "Further Normalization of the Data Base Relational Model", Data Base Systems, Courant Computer Science Symposium 6, Prentice Hall, Englewood Cliffs, New Jersey, 1972, p. 33-64.

REFERENCES (Concluded)

22. D.C. Tsichritzis and F.H. Lochovsky, Data Management Systems, Academic Press, New York, 1977, Chapters 2 and 7.

23. K.J. Biba, "Integrity Considerations for Secure Computer Systems", ESD-TR-76-372, Electronic Systems Division, AFSC, Hanscom AFB, Massachusetts, April 1977 (ADA039324).